이동 컴퓨팅 환경에서 연속 부분 부합 질의의 효과적인 처리를 위한 캐시 관리 방안

(A Cache Management Scheme for Effective Processing of Continuous Partial Match Queries in Mobile Computing Environments)

정 연 돈 † 이 지 연 †† 이 윤 준 ††† 김 명 호 †††
(Yon Dohn Chung) (Ji Yeon Lee) (Yoon Joon Lee) (Myoung Ho Kim)

요 약 본 논문은 이동 컴퓨팅 환경에서 연속 부분 부합 질의의 효과적인 처리를 위한 캐시 관리 방안을 제안한다. 연속 부분 부합 질의란 질의의 결과가 클라이언트의 메모리에 일관성을 유지하면서 지속되는 부분 부합 질의이다. 기존의 이동 환경을 위한 캐시 관리 기법은 레코드 식별자를 기반으로 하는 방법들이다. 하지만, 부분 부합 질의는 데이타의 내용을 기반으로 탐색하는 질의이기 때문에 이러한 레코드 식별자를 기반으로 하는 방법들은 캐시 관리를 효율적으로 할 수 없다. 제안하는 캐시 관리 방안에서는, 이동 클라이언트의 캐시 상태를 프레디킷(predicate)으로 기술하고, 서버가 캐시 관리를 위해 클라이언트에게 방송하는 캐시 무효화 정보, 즉 Cache Invalidation Reports (CIR)을 프레디킷으로 구성한다. 이러한 프레디킷 표현을 사용하여, 일련의 캐시 관리 기법-억지 방법 (the brute-force method), 빼기 방법 (the subtraction method), 교차 방법 (the intersection method)-들을 제안한다. 그리고, 제안하는 방법의 계산 복잡도를 계산한다.

Abstract This paper proposes a cache management scheme for effective processing of continuous partial match queries in mobile computing environments. The continuous partial match query is a partial match query whose result continues to exist consistently in the client's memory. Conventional cache management methods for mobile environments are record ID-based ones. However, since the partial match query is a content-based retrieval, the record ID-based approach does not manage the cache efficiently. In our proposed scheme, we represent the cache state of a mobile client as a predicate, and construct the cache invalidation reports (which the server broadcasts to clients for cache management) with predicates. Using the predicate representation, we also propose a set of cache management methods: the brute-force method, the subtraction method, and the intersection method. And, we analyze the computational complexity of the proposed methods.

1. 서 론

무선 통신 및 컴퓨터 기술의 발달에 따라 사용자가 이동하면서 정보를 검색, 처리할 수 있도록 하는 이동 컴퓨팅(mobile computing)이라는 환경이 새로이 생겨나고 있다[1]. 그림 1은 본 논문에서 가정하고 있는 이동 컴퓨팅 시스템 구조로서, 유선 네트워크에 연결된 서버 컴퓨터와 다수의 이동 클라이언트(mobile client)들이 무선 통신망으로 연결되어 있다[1, 2]. 무선 통신망을 구성하는 세부적인 기술은 CDPD, PCS, Radio 통신 등의 무선 통신 방법을 사용할 수 있다. 본 논문이 기술하는 내용은 특정 무선 통신 기술을 전제로 하지 않는다.

이동 컴퓨팅 시스템이란 기본적으로 기존의 분산 컴퓨팅 시스템에서 유선 네트워크로 구축되어 있던 컴퓨터들을 무선 통신 네트워크로 연결한 것으로 생각할 수

• 본 연구는 BK21 산학협력자금의 부분적인 지원을 받았음.
† 비 회 원 : 한국과학기술원 정보전자연구소 연구원
ydchung@dbserver.kaist.ac.kr
†† 비 회 원 : 현대정보기술(주) 기술센터 미래기술팀 연구원
jylee@hit.co.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
yjlee@dbserver.kaist.ac.kr
mhkim@dbserver.kaist.ac.kr
논문접수 : 2000년 9월 6일
심사완료 : 2001년 3월 22일

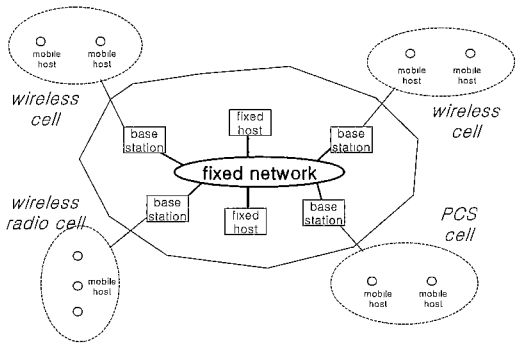


그림 1 이동 컴퓨팅 시스템 구조

있다. 하지만, 유선 네트워크와 무선 네트워크의 차이는 기존의 분산 시스템 환경과 매우 다른 특성 및 응용들을 이동 컴퓨팅 환경에 부과하고 있다[1, 2]. 먼저, 사용자가 지니는 컴퓨터 단말기가 휴대 가능하여야 한다는 점이다. 따라서 기존의 분산 시스템에서의 일반 컴퓨터와 비교하여 단순한 기능만을 처리할 수 있는 소형의 기기일 수밖에 없다. 또한 무선 통신의 특성상 지리적 혹은 물리적인 이유로 서버와 클라이언트 사이의 통신이 비교적 자주 단절되며, 통신 대역폭 역시 유선 통신에 비하여 현저히 작다.

이러한 이동 컴퓨팅 환경이 지니는 특성들로 인하여 서버로부터 다수의 이동 클라이언트를 대상으로 데이타를 방송하는 기법과, 클라이언트에서 서버가 전달한 데이타들을 자체적으로 캐시(cache)하여 사용하는 방법들에 대한 연구가 있었다[2, 3]. 이동 클라이언트에서 데이타 캐싱은 무선 통신이 지니는 특성에서 볼 때, 통신 비용 및 에너지 사용량을 효과적으로 줄이는 방법이다. 클라이언트에서 서버로 데이타를 요청하는 과정 없이 자신이 캐시한 데이타를 사용할 수 있기 때문에, 이에 필요한 통신비용을 줄이면서 휴대 단말기에서 사용하는 전지의 에너지 소모를 줄일 수 있기 때문이다[2, 3].

대부분의 기존 연구들에서 고려한 질의의 유형은 레코드의 식별자(identifier) 혹은 기본 키(primary key)를 기반으로 하고 있다. 본 논문에서는 기본 키가 아닌 하나 이상의 애트리뷰트들을 사용하여 데이타를 검색하는 부분 부합 질의(continuous partial match query)를[4, 5, 8] 다루고자한다. 가령, 주식 정보를 나타내는 테이블(R)을 구성하는 애트리뷰트들이 '회사명', '가격', '매수량', '매도량' 이라고 할 때, '가격' 애트리뷰트와 '매수량' 애트리뷰트 만을 지정하여 검색하는 다음과 같은 질의가 그 예가 될 수 있다.

select *
from *R*
where *'가격<30000 and 5000<'매수량<10000*

연속(continuous) 질의란 질의가 처리된 후 그 결과를 일회성으로 사용하는 것이 아니라, 일정기간 계속하여 사용자에게 결과를 제공하는 것이다[6, 7]. 물론 이 기간동안 데이타의 값이 변함에 따라 질의의 결과 역시 항상 최근의 결과를 반영하여야 한다. 주식 정보의 예를 들어보면, 특정 기업의 주식 레코드 값을 항상 단말기 화면에 나타나도록 하는 질의를 연속 질의라고 할 수 있다. 본 논문에서 고려하는 연속 부분 부합 질의란 부분 부합 질의의 형태가 연속 질의로 사용되는 것을 말한다. 위에서 SQL문으로 예를 든 부분 부합 질의의 결과를 사용자가 일정 시간 동안 자신의 단말기 상에 표시되도록 하는 경우가 이에 해당된다.

이동 컴퓨팅 환경에서 이동 사용자가 서버에게 위와 같은 연속 부분 부합 질의를 제시할 경우, 서버는 이에 해당하는 데이타 레코드들을 사용자에게 전달하게 되며 사용자는 이 데이타들을 캐시에 저장하여 일정기간 동안 보관하게 된다. 이 기간동안 데이타의 변화에 따라 해당 질의의 결과가 변하는 경우에는 사용자의 캐시에 저장되어 있는 결과 역시 최신의 결과로 갱신되어야 한다. 이를 위해 서버와 이동 클라이언트는 캐시 데이타의 일관성 유지를 위한 노력을 하게된다.

본 논문에서는 이동 클라이언트들의 연속 부분 부합 질의를 위한 캐시 일관성 유지 기법으로 캐시 무효화 방송(cache invalidation report) 기법을 사용한다. 캐시 무효화 기법이란 이동 클라이언트들이 캐시하고 있는 데이타들의 일관성 유지를 위하여 서버가 모든 데이타들의 변화 정보를 방송(broadcasting)으로 전달하는 방법이다[2]. 하지만, 연속 부분 부합 질의는 내용 기반 검색 질의이기 때문에 기존의 데이타별 변화 정보를 사용하는 캐시 무효화 방법은 정확성 및 효율성 측면에서 바람직하지 못하다. 따라서, 본 논문에서는 내용기반 질의인 연속 부분 부합 질의를 위하여 프레디킷 기반 캐시 관리 기법을 제안한다. 부분 부합 질의를 다중 애트리뷰트 해싱(multiattribute hashing)[8] 기법을 이용하여 프레디킷으로 기술하고, 사용자의 캐시에 저장되어 있는 부분 부합 질의를 기술하는 프레디킷과 서버가 방송으로 전달하는 프레디킷을 이용하여 이동 클라이언트의 캐시 일관성을 유지하는 방법이다.

논문의 구성은 다음과 같다. 다음 장에서 이동 컴퓨팅 시스템에서 데이타 캐싱이 지니는 중요성과 캐시 일

관성 기법인 캐시 무효화 기법, 그리고 본 논문이 고려하는 연속 부분 부합 질의에 대하여 알아본다. 3장에서는 본 논문이 제안하는 프레디킷 기반 캐싱 기법을 소개한다. 4장에서 제안하는 기법의 사용시 캐시 일관성 유지를 위하여 클라이언트에서 서버로 요청하는 나머지 프레디킷의 생성 기법에 대하여 알아보고, 5장에서 결론을 맺는다.

2. 배경

2.1 이동 클라이언트에서의 데이타 캐싱

이동 컴퓨팅 환경이 지니는 특징 중에서 무선 통신을 사용한다는 점과 휴대용 단말기를 사용한다는 것은 기존 분산 컴퓨팅 환경과 다른 몇 가지 제약을 제공한다. 무선 통신의 경우 유선 통신에 비하여 통신 대역폭이 협소하며, 또한 동시에 설정할 수 있는 연결의 수가 제한되어 있다는 점이다. 따라서, 가능한 서버와의 통신 횟수 및 데이타 전송량을 줄이는 방법을 필요로 한다. 또한, 휴대용 단말기를 사용하기 때문에 전지를 에너지원으로 사용하게 되며, 제한된 에너지를 효과적으로 사용해야 한다. 무선 통신의 특성상 데이타의 전송에 필요한 에너지량이 데이타 수신에 필요한 에너지량에 비해 1000배 가량 크다고 알려져 있다[2]. 이러한 이동 컴퓨팅 환경의 제약들로 인하여 이동 클라이언트에서 데이타를 캐시하여 사용하는 기법이 많이 연구되고 있다.

데이타 캐싱이란 이동 클라이언트가 서버로부터 전달받은 데이타를 자신의 캐시메모리에 저장하여, 향후 이 데이타를 다시 필요로 할 경우 서버에게 데이타를 요청하지 않고 자신의 메모리에서 인출하여 사용하는 방법이다. 데이타 캐싱을 사용할 경우 가장 중요한 문제는 서버에 존재하는 데이타와 이동 클라이언트가 캐시하고 있는 데이타들 사이의 일관성을 유지하는 것이다. 기존의 분산 시스템에서도 다양한 캐시 일관성 유지 기법들이 사용되었지만, 이동 컴퓨팅 환경에서 이 방법들을 사용하기에는 효율성 측면에서 부적합하다. 기존의 유선 통신과 이동 컴퓨팅 환경에서의 무선 통신이 갖는 차이점 때문이다.

이동 컴퓨팅 환경에서 이동 클라이언트들의 캐시 일관성을 유지하는 방법으로 많이 사용되는 방법으로 캐시 무효화 방송(cache invalidation report broadcasting) 기법이 있다. 서버가 데이타들의 변화 정보를 방송 기법으로 모든 클라이언트들에게 전송하는 방법이다. 방송 기법을 이용하기 때문에, 제한된 무선 통신 대역을 다수의 클라이언트들이 공유하여 사용함으로써 주파수의 효율적 사용이 가능하며, 서버에게 개별적으로 요청하지 않고 서버가 전달하는 정보를 수신만 하면 되기 때문에 에너지 효율성 측면에서도 뛰어난 방법이다. 각 클라이언트들은 이 방송을 수신하여 자신이 캐시하고 있는 데이타들 중에서 변화한 것이 있는지를 확인하게 되며, 변화된 데이타에 대하여 서버에게 최신의 값을 다시 요청하게 된다. 캐시 무효화에 이용되는 방법들 중에서, 본 논문에서 사용하는 방법은 각 데이타별로 가장 최근 갱신 시간을 방송하는 타임스탬프(timestamp) 기법이다[3]. 각 데이타별로 변화 유무만을 전달하는 AT (amnesic terminal) 방법에 비해, 이동 클라이언트들의 통신 단절이 빈번한 무선 통신 환경에 적합한 방법이다[3].

2.2 연속 부분 부합 질의

부분 부합 질의란 데이타 레코드를 구성하는 애트리뷰트들 중에서 일부 애트리뷰트의 값을 지정하여 검색하는 내용 기반 검색 질의로[5, 8, 9], 다양한 응용들에서 널리 사용되는 질의 형태이다. 이동 컴퓨팅 환경의 응용시스템 중 하나인 무선 정보 시스템을 예로 부분 부합 질의에 대하여 살펴보도록 하자. 서버가 실시간 주식 정보를 제공하고 이동 클라이언트들은 휴대용 단말기를 사용하여 자신이 관심을 갖는 주식들에 대하여 질의 검색을 한다고 가정한다. 각 사용자들이 자신이 보유하고 있는 특정 회사의 주식 정보를 검색하기도 하지만, 많은 경우 주식의 가격 및 매수/매도 동향 등에 대하여 관심을 가질 수 있다. 이러한 경우 다음과 같은 부분 부합 질의 형태로 서버에게 검색을 요청하게 된다.

```
Q1 : select   *
     from     R
     where    20,000 ≤ '가격' < 30,000
                and '매수량' < 10,000
Q2 : select   *
     from     R
     where    '업종'='전자 or 통신 or 반도체
     and      '가격' < 100,000
```

일반적인 질의들이 클라이언트가 결과를 받은 후 사용자에게 전달되는 것으로 질의의 생명이 끝나는 것과 달리, 연속 질의란 사용자가 질의를 제거하지 않는 한 그 결과가 계속 지속하여 사용자에게 전달되도록 하는 질의를 말한다[6, 7]. 다음은 연속 질의의 처리를 나타내는 알고리즘이다.

```
FOREVER DO
        EXECUTE Q;
        Return its result to user;
END DO
```

그림 2 연속 질의 처리 알고리즘

연속 부분 부합 질의란 연속 질의가 부분 부합 질의의 형태를 띄는 것을 말한다. 가령, 어떤 이동 사용자가 위에서 예로 든 Q1이나 Q2와 같은 부분 부합 질의의 결과가 자신의 이동 단말기 화면에 항상 나타나도록 질의를 만든 것이 연속 부분 부합 질의가 된다. 물론, 단말기의 화면에 나타나는 기간동안 항상 최신의 정보로 갱신되어야 한다. 본 논문에서 다루고자 하는 내용은, 무선 정보 시스템 환경에서 이동 사용자가 연속 부분 부합 질의를 사용하는 경우, 이동 사용자들의 캐시 일관성을 유지하기 위한 방법이다.

3. 프레디킷 기반 캐싱

3.1 기존 방법의 문제점

기존의 레코드 식별자를 사용하는 캐시 관리 기법들은 서버가 방송으로 전달하는 캐시 무효화 정보를 통해, 현재 이동 클라이언트 자신의 캐시에 저장되어 있는 데이타 레코드들의 변화 유무를 판단하였다. 가령, 캐시 무효화 정보에서 식별자가 10인 레코드가 서버에서 갱신되었다는 정보를 통해, 자신의 캐시에 저장되어 있는 10번 레코드의 캐시 일관성이 보장되지 않는다는 사실을 발견하여, 서버에게 10번 데이타 레코드의 새로운 값을 요청하는 것이다. 하지만, 이러한 레코드 식별자를 기반으로 하는 캐시 관리 방법들은 본 논문에서 고려하고 있는, 부분 부합 질의와 같은 내용 기반 검색 질의에는 사용할 수 없다. 그 이유는 다음과 같은 상황이 발생하기 때문이다.

- 어떤 이동 클라이언트가 주가가 10000원 이하인 주식 레코드를 자신의 캐시에 저장하고 있다고 가정하자. 시간이 지남에 따라, 이전에 10500원이던 어떤 주식 레코드가 가격이 내려 9900원으로 바뀌었다고 하면, 이 주식 레코드는 이전에는 이 이동 클라이언트의 캐시에 저장되어 있지 않았지만, 지금은 저장되어야 하는 레코드가 된다. 하지만, 레코드 식별자를 기반으로 하는 캐시 관리 방법은 현재 캐시되어 있는 데이타들의 변화 유무만을 조사하기 때문에, 이러한 상황을 처리하지 못하게 된다.
- 어떤 이동 클라이언트가 주가가 10000원 이하인 주식 레코드를 자신의 캐시에 저장하고 있다고 가정하자. 시간이 지남에 따라, 이전에 9900원이던 어떤 주식 레코드가 가격이 올라 10100원으로 바뀌었다고 하면, 이 주식 레코드는 이전에는 이 이동 클라이언트의 캐시에 저장되어 있었지만, 지금은 저장되지 말아야 하는 레코드가 된다. 기존의 레코드 식별자를 기반으로 하는 캐시 관리 방법에서는, 현재 캐시되어 있는 이 레코드가 변화하였다는 사실을 서버가 방송으로 전달하는 정보를 사용하여 알게된 후, 서버에게 새로운 값을 전달받게 된다. 그런 다음, 이 데이타의 바뀐 값을 확인하여 자신이 필요로 하는 레코드가 아님을 알게되고, 자신의 캐시에서 제거하게 된다. 이 과정에는 필요치 않은 데이타 통신 비용이 소모된다.

이러한 문제점들을 해결하기 위하여, 본 논문에서는 서버가 클라이언트들에게 방송으로 전달하는 캐시 무효화 정보를 프레디킷으로 구성하는 캐시 관리 방법을 제안한다.

3.2 프레디킷 표현

부분 부합 질의는 데이타 레코드의 내용을 지정하여 검색하는 내용 기반 질의이기 때문에, 본 논문에서는 각 데이타 레코드들이 지니는 데이타 값들을 효과적으로 표현하기 위하여, 다중 애트리뷰트 해싱을 통한 프레디킷(predicate)을 사용하였다. 데이타 레코드를 기술하는 프레디킷을 '레코드 프레디킷'라고 부르고, 질의를 기술하는 프레디킷을 '질의 프레디킷'이라고 부르기로 한다.

'레코드 프레디킷'이란 하나의 데이타 레코드를, 레코드를 구성하는 각 애트리뷰트별로 정의된 해시 함수에 따라 이진 비트로 표시한 후, 그 결과를 결합한 비트열을 말한다. 각 애트리뷰트별 해시 함수는 애트리뷰트의 특성에 따라 효과적으로 선정되어 주어진다고 가정하며, 해시 결과는 '0' 또는 '1'의 배열, 즉 비트 열(bit stream)이라고 가정한다.

정의 1. 주어진 레코드 R_k를 구성하는 애트리뷰트 A_1, A_2, ..., A_n에 대하여 각 애트리뷰트에 대한 해시 함수 H_i를 사용하여 해시한 결과 비트열을 B_i라고 하자. R_k의 레코드 프레디킷 RP_k는 $B_1+B_2+...+B_n$ 이다. 여기서 '+'는 비트열의 결합(concatenation) 연산자이다. ■

'질의 프레디킷'이란 주어진 부분 부합 질의에서 값을 지정하는 애트리뷰트에 대하여 해당 해시 함수를 사용한 결과 비트열을, 지정되지 않은 애트리뷰트에 대해서는 해시 결과 비트열의 크기에 해당하는 '*' 문자열을 설정한 후, 이들을 결합한 배열을 나타낸다.

정의 2. 주어진 부분 부합 질의 Q_k에서 지정한 애트

리뷰트 A_i에 대하여 해시 함수 H_i를 적용한 결과 비트열을 B_i라고 하고, Q_k에서 지정하지 않은 애트리뷰트 A_j에 대해서는 H_j를 통해 생성되는 비트열의 크기만큼 '*' 문자로 이루어진 배열 B_j를 생성한다. Q_k의 질의 프레디킷인 QP_k는 $B_1+B_2+...+B_n$이다. ■

주식 정보를 제공하는 응용 시스템의 예를 사용하여 '레코드 프레디킷'과 '질의 프레디킷'의 예를 설명하도록 하자. 시스템에서 사용하는 주식 정보 레코드는 다음의 4개 애트리뷰트로 구성되어 있다. A_1='종목번호', A_2='매도량', A_3='매수량', A_4='현재가격'. 그리고 각 애트리뷰트별로 다음과 같은 해시 함수가 사용된다고 가정한다.

$$h_{\text{종목번호}}(x)=\begin{cases}00 \text{ if } x<100\\01 \text{ if } 100\le x<200\\10 \text{ if } 200\le x<300\\11 \text{ if } 300\le x\end{cases}$$

$$h_{\text{매도량}}(x)=\begin{cases}00 \text{ if } x<10000\\01 \text{ if } 10000\le x<20000\\10 \text{ if } 20000\le x<30000\\11 \text{ if } 30000\le x\end{cases}$$

$$h_{\text{매수량}}(x)=\begin{cases}00 \text{ if } x<10000\\01 \text{ if } 10000\le x<20000\\10 \text{ if } 20000\le x<30000\\11 \text{ if } 30000\le x\end{cases}$$

$$h_{\text{현재가격}}(x)=\begin{cases}000 \text{ if } x<5000\\001 \text{ if } 5000\le x<10000\\010 \text{ if } 10000\le x<20000\\011 \text{ if } 20000\le x<30000\\100 \text{ if } 30000\le x<40000\\101 \text{ if } 40000\le x<50000\\110 \text{ if } 50000\le x<60000\\111 \text{ if } 60000\le x\end{cases}$$

위 해시 함수들을 이용하여 레코드 프레디킷을 만들어 보자. 어떤 레코드의 내용이 [종목번호=150, 매도량=13,000, 매수량=2,000, 현재가격=22,000]이라고 하면, 이 레코드의 프레디킷은 '010100011'이 된다. 마찬가지로 앞장의 Q1과 같은 부분 부합 질의가 있을 때, Q1의 질의 프레디킷은 '****00011'이 된다. 해시 함수에 의해 애트리뷰트 A_i가 l_i 길이의 비트 열로 나타내어질 경우, 프레디킷의 크기 N은 $\sum_{i=1}^{n} l_i$ 가 된다(n은 애트리뷰트의 개수이다). 애트리뷰트별 해시 비트열의 적정 크기 및 프레디킷 구성 시 해시된 비트 열의 결합 순서 관한 연구 결과[9]를 이용할 수 있다.

3.3 프레디킷 기반 캐싱 및 캐시 무효화 방송

지금부터 이동 클라이언트가 프레디킷을 사용하여 연속 부분 부합 질의를 사용하는 환경을 고려해보자. 이동 클라이언트들은 프레디킷을 사용하여 서버에게 질의를 요청하게 되며, 서버는 이 질의를 처리하여 결과 데이타 레코드들을 클라이언트에게 전달한다. 그러면, 클라이언트는 이 데이타 레코드들을 자신의 캐시에 저장해 두면서, 계속적으로 사용자에게 결과를 알려주게 된다. 이 과정에서 서버에서 데이타의 변화가 발생할 경우 클라이언트의 캐시에 저장되어있는 데이타 역시 갱신되어야 할 것이다. 하지만, 서버가 모든 클라이언트들의 캐시 정보를 유지하는 방법은 서버 확장성(scalability) 측면에서 비현실적이다. 또한, 각 클라이언트가 자신의 캐시 일관성 유지를 위하여 주기적으로 서버에게 메시지를 전송하여 확인하는 방법 역시 무선 통신의 비용 면에서 불가능하다. 본 장에서는 서버가 모든 이동 클라이언트들에게 데이타 레코드들의 변화 정보를 방송으로 전달하는 캐시 무효화 방송 기법을 기반으로 하는 연속 부분 부합 질의의 캐싱 기법을 제안한다.

프레디킷 기반 캐싱

이동 클라이언트들은 자신의 캐시에 저장되어 있는 데이타 레코드들을 기술하는 프레디킷을 가지고 있다. 이는 캐시되어 있는 데이타들이 만족하는 프레디킷으로, 처음에 서버에게 요청한 연속 부분 부합 질의의 프레디킷에 해당한다. 가령, 어떤 클라이언트가 자신의 캐시에 '****00011'이라는 프레디킷을 가지고 있다면, 이것은 캐시에 이 프레디킷을 만족하는 데이타 레코드들이 저장되어 있다는 의미이다. 앞으로 설명의 편의를 위하여 클라이언트가 캐시하는 프레디킷은 하나라고 가정하기로 한다. 클라이언트가 캐시하는 프레디킷은 하나 이상이 있을 수 있으며, 이 경우에도 제안하는 방법은 쉽게 확장 가능하다.

프레디킷 기반 캐시 무효화 방송

본 논문에서 제안하는 방법은 프레디킷을 사용하여 CIR을 구성하는 방법이다. 프레디킷로 이루어진 CIR을 P-CIR이라고 부르기로 한다. 서버가 프레디킷들과 각 프레디킷별 타임스탬프를 캐시 무효화 정보로 방송하고, 클라이언트는 자신의 프레디킷과 서버에서 방송하는 P-CIR에 있는 프레디킷을 비교하여 자신이 저장하고 있는 데이타들이 변화하였는지를 판별하고, 변화하였을 경우 새로운 데이타를 서버에게 요청하는 방법이다

프레디킷 래티스

서버가 제공하는 모든 데이타 레코드들은 해시 함수가 생성하는 비트 열을 통해 프레디킷으로 기술할 수 있다. 존재할 수 있는 모든 프레디킷들 사이의 관계를 나타내기 위하여 '프레디킷 래티스'라는 구조를 정의한다. (그림 3 참조)

정의 3. N개의 비트로 구성된 프레디킷의 프레디킷 래티스(lattice)는 다음과 같이 정의된다.

1. N+2개의 레벨을 갖는 래티스이다.
2. 래티스를 구성하는 각 노드는 N 비트의 프레디킷이며, 각 노드에는 해당 프레디킷의 가장 최근 타임스탬프(즉, 갱신 시각)가 기록된다
3. 최상위 루트 레벨(level 0)을 구성하는 프레디킷은 N개의 비트가 모두 '*'로 구성되며, N번째 레벨을 구성하는 프레디킷은 N개의 비트가 모두 '*'이 아닌 '0' 또는 '1'로 구성된다. (N+1)번째 레벨은 래티스 구성요건을 만족시키기 위하여 인위적으로 설정한 null 노드이다.
4. h번째 레벨에 있는 프레디킷들은 (N-h)개의 '*' 문자와 h개의 '0' 또는 '1'을 갖는다.
5. h번째 레벨에 있는 노드들은 (h+1)번째 레벨에 있는 2N-h개의 자식 노드들에 대한 에지를 갖는다. 에지로 연결되어 있는 두 노드 사이에는 h 레벨의 부모 노드의 h개의 '*'이 아닌 비트가 (h+1) 레벨의 자식 노드에도 동일하게 나타나는 관계를 갖는다. 따라서, 자식 노드의 프레디킷을 만족하는 데이타 레코드들은 항상 부모 노드의 프레디킷을 만족하게 된다.
6. 한 노드가 갖는 타임스탬프는 자식 노드들의 타임스탬프 값들 중에서 가장 큰 값을 갖는다(즉, 가장 최근 시간 값을 갖는다). N번째 레벨 노드는 해당 프레디킷을 만족하는 데이타에 대한 타임스탬프를 갖는다. 데이타의 변화에 따라 프레디킷 래티스를 구성하는 각 노드들의 타임스탬프는 갱신된다. ■

정의 4. 두 개의 프레디킷 P_1과 P_2 사이에는 다음의 3가지 관계가 존재한다.

1. P1을 만족하는 모든 데이타 레코드들이 P2를 만족하는 경우 P1은 P2에 '포함'된다고 한다.
2. P1을 만족하는 데이타 레코드들의 집합과 P2를 만족하는 데이타 레코드들의 집합이 서로 소인 경우 P1과 P2는 서로 '무관'하다고 한다.
3. P1과 P2가 서로 무관하지도 않고 포함 관계에 있지도 않으면, 두 프레디킷은 서로 '교차' 관계이다.

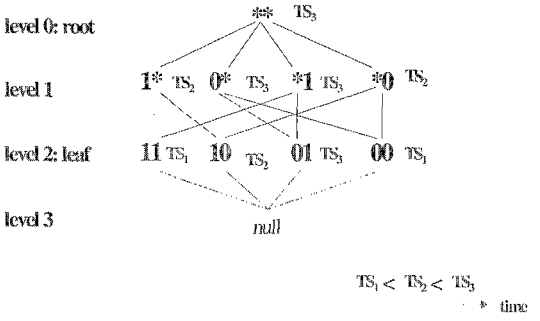


그림 3 2비트 프레디킷 래티스의 예

예제 1. 프레디킷들 사이의 관계에 대하여 다음의 프레디킷들을 사용하여 알아보자.

P_1 = 0011*0**, P_2 = 00**11**, P_3 = **11****

```
PROCEDURE Cache_Management( )
{
        while (true) {
                go to the root node of the P-CIR (i.e., predicate-lattice);
                node = read_a_node( );
                TimeStamp_Check(node);
        } /* go to the next P-CIR */
}

// Notation:
//      TS_cache: the timestamp of cached predicate
//      TS_node: the timestamp of the node currently read from the P-CIR

PROCEDURE TimeStamp_Check(node)
{
        if (TS_cache < TS_node) {
                /* follow the node's edges to find the cache predicate */
                for each relevant child node node_C
                        do TimeStamp_Check(node_C);
        } else {
                /* the cached data covered by this node is valid; do nothing */
                break ;
        }
}
```

그림 4 P-CIR을 이용한 이동 클라이언트의 캐시 일관성 유지 알고리즘

P_1과 P_2는 서로 무관하며, P_1은 P_3에 포함되며, P_2와 P_3는 교차 관계에 있다. ■

캐시 일관성 유지 방법

서버는 모든 클라이언트들에게 P-CIR을 방송하며, 각 클라이언트는 자신이 캐시하고 있는 데이타에 대한 프레디킷에 대한 변화 유무를 방송으로 전달되는 P-CIR에서의 해당 프레디킷의 타임스탬프를 읽어 판단할 수 있다. P-CIR의 구성이 프레디킷 래티스를 너비 우선 탐색(breadth-first search)을 통해 이루어지고, 각 부모 노드에서 자식 노드들로의 에지 정보가 P-CIR에서 유지되도록 방송된다고 가정할 때, 클라이언트의 캐시 일관성 유지 알고리즘은 그림 4와 같다.

3.4 프레디킷 선택 및 나머지 프레디킷

프레디킷 선택

앞장에서 프레디킷 래티스를 사용하여 P-CIR을 구성하는 방법을 설명하였다. 프레디킷 래티스를 사용하여 P-CIR을 구성하는 것은 3^N 개의 프레디킷을 P-CIR로 방송한다는 것을 말한다(여기서 N은 프레디킷의 크기, 즉 비트의 개수이다). 하지만, 데이타 레코드를 구성하는 애트리뷰트의 수가 증가하고, 그에 따른 해시 비트의 수가 커지게 되면 N 값이 커지게 되므로, 3^N이라는 프레디킷의 개수는 캐시 무효화 방법을 사용하는데 있어 큰 문제가 된다. 캐시 무효화 방송에 사용되는 정보의 양이 커지게 되면, 캐시 무효화 방송 주기가 길어지며, 따라서 신속한 캐시 일관성 유지가 불가능해지기 때문이다. 이를 해결하기 위하여 프레디킷 래티스에서 일부 래티스 집합을 선택하여 P-CIR로 구성하는 것을 '프레디킷 선택' 문제라고 정의한다.

프레디킷 선택을 통해 방송되는 프레디킷의 개수는 응용에 따라 적절한 캐시 무효화 주기의 크기를 설정한 후, 그 주기에 방송 가능한 데이타의 양을 계산하여 선정할 수 있다. 본 논문에서는 시스템에서 적정 프레디킷의 개수가 파러미터로 주어진다고 가정한다. 한편, 프레디킷 선택 과정에는 다음과 같은 조건이 만족한다고 가정한다. 캐시 무효화 방송을 통해 이동 클라이언트들에게 전달되는 프레디킷들을 통하여 어떠한 데이타 레코드도 데이타 변화 유무를 판단할 수 있어야 함을 나타낸다.

완전성 조건: 프레디킷 래티스에서 P-CIR 구성을 위하여 프레디킷을 선택할 때, 모든 데이타 레코드가 선택되어진 프레디킷들 중 적어도 하나에 포함되어야 한다. 이를 프레디킷 선택의 '완전성 조건'(completeness requirement)이라고 부르기로 한다.

예제 2. 다음과 같이 프레디킷 선택에 의해 P-CIR로 구성되는 프레디킷 집합들이 있을 때, 각 집합이 완전성 조건을 만족하는지 알아보자. 편의상 3 비트 프레디킷을 사용하기로 한다.

집합 1= { 00*, 01*, 10*, 11* },
집합 2 = { 100, 101, 1**, 0** },
집합 3 = { 10*, 1*1, 0*1, *11 },
집합 4 = { *00, 1** }

집합 1, 집합 2, 집합 4는 완전성 조건을 만족하지만, 집합 3은 완전성 조건을 만족하지 않는다. 집합 3의 프레디킷들로 P-CIR을 구성할 경우, '000' 이나 '110' 등의 프레디킷에 해당하는 데이타 레코드들은 일관성 확인을 할 수가 없다. ■

나머지 프레디킷

그림 4에서 기술한 클라이언트 캐시 관리 알고리즘은 프레디킷 래티스에 존재하는 모든 프레디킷들을 서버가 방송하는 경우에 사용되는 방법이다. 하지만, 프레디킷 래티스 전체가 아닌 일부를 P-CIR로 방송하게 될 경우, 방송되는 P-CIR에서 클라이언트가 캐시하고 있는 프레디킷과 정확하게 일치하는 프레디킷을 찾지 못하는 경우가 생기게 된다. 가령, 위 예제 2의 집합 2와 같이 프레디킷들을 선택하여 방송하는 경우, 클라이언트가 '1*1' 프레디킷을 캐시하고 있는 경우를 살펴보자. ('1*1' 에 해당하는 연속 부분 부합 질의의 결과를 캐시하고 있다는 의미이다.) 서버에서 보낸 정보는 4개의 프레디킷과, 이 프레디킷들의 타임스탬프들이다. 클라이언트가 캐시하고 있는 프레디킷의 타임스탬프가 위 프레디킷들의 타임스탬프가 나타내는 시각과 같다면, 이 클라이언트의 캐시는 최신의 정보를 가지고 있다고 볼 수 있으므로 캐시 갱신 작업이 필요 없다. 하지만, 클라이언트가 가지고 있는 타임스탬프가 아주 오래 전의 것이라면, 적절한 갱신 작업을 필요로 한다. 먼저, 서버가 보내는 프레디킷 '0**'에 대하여 살펴보자. 이 프레디킷이 포함하는 데이타 레코드들은 '000', '001', '010', '011' 프리디킷들에 해당하는 데이타 레코드가 된다. 클라이언트가 캐시하고 있는 '1*1' 프레디킷에 해당하는 데이타 레코드들은 이 '0**' 프레디킷과 '무관'하기 때문에 캐시 일관성 확인 과정에 전혀 도움을 주지 못한다는 점을 알 수 있다. '100' 역시 캐시되어 있는 프레디킷과 무관하다. 한편, '1**' 프레디킷은 클라이언트가 캐시하고 있는 '1*1' 프레디킷에 대한 데이타 레코드들을 '포함'하고 있기 때문에 캐시 일관성 확인 과정에 사용될 수 있다. 또한 '101' 프레디킷은 캐시하고 있는 데이타 레코드의 일부를 나타낼 수 있기 때문에 캐시 일관성 확인에 사용될 수 있다. '1**' 프레디킷의 타임스탬프 값이 클라이

언트가 캐시하고 있는 프레디킷의 그것보다 최근의 값을 갖고 있다면, 클라이언트가 캐시하고 있는 '1*1' 프레디킷에 해당하는 데이타 레코드의 값들 중 변화가 있는 것들이 있을 수 있다는 의미이다. 이때 '101' 프레디킷의 타임스탬프는 캐시의 타임스탬프와 같은 값을 갖는다고 하자. 그러면, '1*1'을 캐시하고 있는 클라이언트 입장에서 '111'에 해당하는 데이타만이 변화하였다는 사실을 알 수 있다. 따라서, 서버에게 '111' 프레디킷을 부분 부합 질의 형태로 전달하여 해당되는 최신의 데이타 레코드들을 다시 받아 캐시 일관성을 유지할 수 있게된다. 이렇게, 캐시 일관성 유지를 위하여 클라이언트가 서버에게 요청하는 프레디킷을 나머지 프레디킷(remainder predicate)이라고 한다.

정의 5. P-CIR의 프레디킷들의 타임스탬프가 캐시 프레디킷의 타임스탬프보다 큰 값, 즉 최근의 시각 값을 나타낼 경우, 이 프레디킷을 '무효 프레디킷'(invalid predicate)이라 하고, 그렇지 않은 프레디킷을 '유효 프레디킷'(valid predicate)이라고 한다. ■

P-CIR 프레디킷 집합은 무효 프레디킷들과 유효 프레디킷들로 구성되어 있다. 무효 프레디킷이란 클라이언트가 캐시하고 있는 데이타 레코드들 중에서 서버 측에서 변화가 발생하여 클라이언트의 캐시를 갱신하여야 함을 나타내며, 유효 프레디킷이란 캐시되어 있는 데이타의 내용이 서버가 저장하고 있는 내용과 일치한다는 의미이다. 물론, 무효 및 유효의 판단은 각 클라이언트가 캐시하고 있는 프레디킷의 타임스탬프에 따라 개별적으로 이루어진다. 캐시되어 있는 데이타에 대한 프레디킷과 무효 및 유효 프레디킷을 비교하여 캐시 내에 저장되어 있는 데이타들 중에서 변화가 생긴 부분만을 찾아내어 서버에게 새로운 데이타를 요구하기 위한 프레디킷이 나머지 프레디킷이다.

4. 나머지 프레디킷 생성 방법

4.1 억지 방법(the brute-force method)

나머지 프레디킷을 찾는 가장 기본적인 방법은 캐시 프레디킷이 나타낼 수 있는 모든 이진 비트열을 생성한 후, 각 이진 비트열이 서버가 보낸 P-CIR 프레디킷들 중에서 어떤 유효 혹은 무효 프레디킷들에 의해 나타내어 지는지를 판단하는 방법이다. 가령, 어떤 클라이언트가 '1*1' 이라는 프레디킷을 캐시하고 있다고 하고, 서버에서 '*00', '*01', '*10', '*11'의 4개 프레디킷을 P-CIR로 전달한다고 가정하자. 이들 프레디킷 중에서 '*00', '*01' 프레디킷은 유효 프레디킷이고, 나머지들은 무효 프레디킷이라고 하면, 이 클라이언트가 캐시 관리를 위해 서버에게 보내는 나머지 프레디킷은 다음과 같이 구해진다.

먼저, 캐시 프레디킷이 나타낼 수 있는 가능한 모든 이진 비트열을 구한다. '101', '111' 이다. 이 두가지 이

```
// Notation:
//   cache_predicate: the predicate describing client's cache,
//   valid_predicate: the valid predicate in the P-CIR,
//   remainder_predicate_set: the set of remainder predicates

PROCEDURE Remainder_Generation_with_Brute_Force (cache_predicate)
{
      remainder_predicate_set = { }; // initially an empty set
      generate all possible binary predicates of cache_predicate;
      for each binary predicate P (in the above) do {
            if ( P is not included by any valid_predicate in the P-CIR) {
                  add P into remainder_predicate_set ;
            } else { break; // do nothing }
      }
}
```

그림 5 억지 방법을 이용한 나머지 프레디킷 생성 알고리즘

진 비트열 중에서 '101' 비트열은 유효 프레디킷인 '*01' 에 포함되며, '111'은 어떤 유효 프레디킷에도 포함되지 않는다. 따라서, 이 클라이언트가 캐시하고 있는 (그리고, 캐시하고 있어야 하는) 데이타 레코드들 중에서 '101'에 해당하는 데이타 레코드들은 캐시의 내용과 서버가 가지고 있는 내용이 일치한다는 점을 알 수 있으며, '111'에 해당하는 데이타만 서버로부터 새롭게 받아와야 한다. 이 경우, '111'이 나머지 프레디킷이 된다. 클라이언트가 캐시하고 있는 프레디킷의 모든 가능한 이진 비트열을 생성하여 나머지 프레디킷을 구하는 방법을 '억지 방법(brute-force method)'이라고 부르기로 한다. 이 방법의 계산 복잡도는 캐시 프레디킷이 가지고 있는 '*'의 개수를 'c'라고 하고, P-CIR로 전달되는 프레디킷의 유효 프레디킷의 개수를 vp라 할 때, $O(vp \times 2^c)$ 이다. 억지 방법을 사용할 경우, 유효 프레디킷만을 사용하여 나머지 프레디킷을 구한다. 유효 프레디킷은 그 프레디킷으로 표현되는 모든 데이타 레코드들이 유효하다는 의미이지만, 무효 프레디킷은 프레디킷 래티스의 정의에서 알 수 있듯이, 그 프레디킷으로 표현되는 데이타들 중에서 변화한 것이 존재한다는 의미이기 때문이다. 즉, 무효 프레디킷으로 표현되는 모든 데이타들이 변화하였다는 의미는 아니다.

4.2 빼기 연산 방법(the subtraction method)

억지 방법의 계산 복잡도는 캐시 프레디킷의 '*' 문자 개수에 따라 거듭 제곱으로 증가하기 때문에, 현실적인 사용이 어렵다. 빼기 연산 방법은 캐시 프레디킷과 유효 프레디킷의 빼기 연산(subtraction)을 통해 나머지 프레디킷을 찾는 방법이다. 가령 캐시 프레디킷이 '1*1'이며, P-CIR의 유효 프레디킷이 '111'이면, 나머지 프레디킷은 '1*1'에서 '111'을 뺀 '101'이 된다. 나머지 프레디킷을 찾기 위한 빼기 연산은 '나머지 프레디킷 = 캐시 프레디킷 - 유효 프레디킷'의 형태로 사용하며, 다음과 같이 정의된다. 그림 6는 빼기 연산 방법을 사용하여 나머지 프레디킷을 생성하는 알고리즘이다.

정의 6. 빼기 연산 '-'은 비트별 연산(bit-wise operation)으로, '나머지 프레디킷=캐시 프레디킷-유효 프레디킷'의 식에서 각 비트별로 다음과 같이 정의된다. 나머지 프레디킷의 비트 값이 '복수'라는 것은 나머지 프레디킷이 여러 개가 존재할 수 있다는 의미이다. '복수' 라는 결과가 둘 이상 나타나면 '복수' 개의 나머지 프레디킷을 찾아야한다. 한 비트에서만 '복수'라는 결과가 나올 경우에는, '복수' 프레디킷이 나오지 않으므로, '0' 또는 '1'로 표에 기술되어 있는 값을 갖는다. 복수 개의 나머지 프레디킷은 억지 방법에서와 같이 모든 이진 비트열을 구하여 계산한다.

캐시 프레디킷의 비트 값	유효 프레디킷의 비트 값	나머지 프레디킷의 비트 값
1	*	1
0	*	0
*	*	*
1	1	1
0	1	0
*	1	0 또는 복수
1	0	1
0	0	0
*	0	1 또는 복수

예제 3. 빼기 연산을 이용한 나머지 프레디킷의 생성 예를 살펴보자

캐시 프레디킷	유효 프레디킷	나머지 프레디킷
11*1	1111	1101
11*1	11**	없음(포함관계)
1***	1*10	1*00, 1*01, 1*11
1*1*	1*0*	1*1*

첫 번째 예는 캐시 프레디킷 '11*1'이 가리키는 데이타 중에서 일부인 '1111'이 유효하기 때문에, 유효 프레디킷이 나타내는 부분을 제외한 나머지인 '1101'이 나머지 프레디킷이 된다. 둘째 예는 캐시 프레디킷이 가리키는 모든 데이타 레코드들이 유효 프레디킷에 포함되는 내용이기 때문에 - 즉, 클라이언트 캐시의 내용이 서버와 일치하므로) 캐시 갱신이 필요없는 경우이다(그림 6의 알고리즘에서 **if** 문장). 세 번째 경우는, 캐시 프레디킷이 나타내는 레코드 중에서 유효 프레디킷으로 일관성이 확인된 부분을 제외한 나머지 부분들이 하나의 프레디킷으로 기술될 수 없는 경우로서, 여러 개의 프레디킷으로 기술된다. 마지막 예는 유효 프레디킷으로 캐시 프레디킷의 일관성 확인을 하지 못하는 경우이다. 따라서, 캐시하고 있는 데이타의 일관성 유지를 위해 캐시 프레디킷이 나머지 프레디킷이 되어 서버에게 전달되도록 해야한다. ■

빼기 연산 방법의 계산 복잡도는 캐시 프레디킷과 유효 프레디킷의 각 비트 조합에 따라 결정된다. 복수 개의 나머지 프레디킷이 생성되지 않는다면 캐시 프레디킷의 개수와 유효 프레디킷의 개수의 곱으로 계산 복잡도가 결정되지만 복수 프레디킷이 발생하는 경우는 캐시 프레디킷이 나타낼 수 있는 모든 이진 비트 열을 생성하는 계산 복잡도를 필요로 하게 된다. 즉, 유효 프레

```
// Notation:
//   cache_predicate: the predicate describing client's cache,
//   valid_predicate: the valid predicate in the P-CIR,
//   remainder_predicate: the remainder predicate

PROCEDURE Remainder_Generation_with_Subtraction (cache_predicate)
{
      remainder_predicate = cache_predicate;
      for each valid_predicate in P-CIR do {
            if ( remainder_predicate is not included by valid_predicate ) {
                  remainder_predicate = remainder_predicate - valid_predicate ;
                  // remainder_predicate이 복수 개가 된 경우,
                  // 각각의 remainder_predicate에 대하여 모두 빼기 연산한다.
            } else { break; // do nothing }
      }
}
```

그림 6 빼기 연산을 이용한 나머지 프레디킷 생성 알고리즘

디킷의 개수를 vp, 캐시 프레디킷에 나타나는 '*' 문자의 개수를 c1, 유효 프레디킷에 나타나는 '*' 문자의 개수를 c2, 복수 개의 나머지 프레디킷이 발생할 확률을 α라고 할 때, 빼기 연산 방법의 계산 복잡도는 $O(\alpha \cdot 2^{c1-c2} \cdot vp + (1-\alpha) \cdot N \cdot vp)$가 된다.

4.3 교차 연산 방법(the intersection method)

빼기 연산 방법을 사용하는 경우 억지 방법에 비하여 계산 복잡도를 상당 부분 줄일 수 있지만, 복수 개의 나머지 프레디킷이 생기는 경우로 인하여, 아직까지 복잡도 측면에서 현실적 사용이 어렵다고 볼 수 있다.

이러한 문제가 발생하는 이유는 무효 프레디킷을 사용하지 못하고, 유효 프레디킷들만을 이용하여야 하는데서 기인한다. 유효 프레디킷은 해당 프레디킷이 나타내는 모든 데이타 레코드들에 대한 캐시 일관성 확인을 보증하지만, 무효 프레디킷의 경우는 해당 프레디킷이 나타내는 데이타 레코드들 중에서 갱신이 필요한 데이타가 하나 이상 존재한다는 점을 나타낸다.

따라서, 유효 프레디킷과 무효 프레디킷에 공통으로 해당되는 부분이 존재할 수 있으며, 이로 인해 유효 프레디킷만을 사용할 수 밖에 없는 것이다. 이 절에서는 P-CIR로 방송되는 모든 프레디킷들이 서로 교집합이 생기지 않도록 구성하여 나머지 프레디킷 생성을 쉽게 하는 교차 연산 방법을 제안한다.

정의 7. 프레디킷의 비트 수가 N일 때, 이 중에서 고정된 l 개의 비트는 '0' 또는 '1'로 기술되고, 나머지 N-l 개의 비트는 '*' 문자로 나타나는 서로 다른 2^l 개의 프레디킷 집합을 1-비트 커버(1-bit cover)라고 한다.

예제 4. 3-비트 프레디킷에서 2-비트 커버는 {00*, 01*, 10*, 11*} 또는 {0*0, 0*1, 1*0, 1*1} 또는 {*00, *01, *10, *11}이다.

관찰 1. l-비트 커버는 '완전성 조건'를 만족한다. ■

교차 방법을 사용하기 위한 P-CIR 프레디킷의 선택 방법은 다음과 같다. 먼저, P-CIR 방송을 위한 프레디킷의 개수가 'K'라고 주어질 때, $j = 2^l < K$ 인 가장 큰 l 값을 구한다.

이 l 값을 이용하여 프레디킷-래티스에서 l-비트 커버를 구한다. (K-j)개의 이진 비트 열(즉, '*' 문자가 없는 비트 열)과 l-비트 커버를 이루는 프레디킷이 P-CIR 프레디킷이 된다.

위와 같은 방법으로 P-CIR이 구성될 경우, 무효 프레디킷과 교차 연산을 사용하여 클라이언트의 캐시 일관성을 확인할 수 있다. 교차 연산은 '나머지 프레디킷 = 캐시 프레디킷 ∩ 무효 프레디킷'의 형태로 사용되며 다음과 같이 정의된다.

정의 8. 교차 연산 '∩'은 비트 별 연산(bit-wise operation)으로서 '나머지 프레디킷 = 캐시 프레디킷 ∩ 무효 프레디킷'에서 캐시 프레디킷과 무효 프레디킷의 각 비트 값에 따라 나머지 프레디킷의 비트 값이 다음과 같다.

캐시 프레디킷의 비트 값	무효 프레디킷의 비트 값	나머지 프레디킷의 비트 값
1	*	1
0	*	0
*	*	*
1	1	1
0	1	없음
*	1	1
1	0	없음
0	0	0
*	0	0

이 테이블에서 나머지 프레디킷의 비트 값이 '없음'을 나타내는 경우는 나머지 프레디킷이 존재하지 않는다는 의미이며, 이는 캐시 프레디킷의 내용 중 무효 프레디킷에 해당되는 데이타가 존재하지 않으므로, 캐시 갱신이 필요 없다는 것을 말한다. 어느 한 비트의 값이 '없음'일 경우, 나머지 프레디킷은 존재하지 않는다.

예제 5. 교차 연산을 이용한 나머지 프레디킷의 예이다. ■

캐시 프레디킷	무효 프레디킷	나머지 프레디킷
11*1	1111	1111
11*1	11**	11*1
1***	1*10	1*10
1*1*	1*0*	없음

그림 7은 교차 연산을 이용하는 나머지 프레디킷 생성 알고리즘이다. *l*-비트 커버를 이루는 무효 프레디킷들을 이용하여 나머지 프레디킷을 생성한 후, '*'이 없는 이진 비트 열 중에서 유효한 것들을 이용하여 생성한 나머지 프레디킷들 중에서 캐시 일관성이 유지되고 있는 부분을 걸러낸다.

교차 연산을 이용하여 나머지 프레디킷을 생성하는 방법은 빼기 연산 방법과 달리, 복수 개의 나머지 프레디킷이 발생하지 않기 때문에, 다음과 같은 계산 복잡도를 갖는다. 프레디킷의 비트 수를 'N', 무효 프레디킷의 개수를 '*ip*'라 할 때, $O(N \cdot ip)$이다(이진 비트 열들을 이용하여 나머지 프레디킷을 정제하는 부분은 전체 복잡도에 미치는 영향이 작기 때문에 생략한다).

5. 비 교

3.1절에서 기술하였듯이, 기존의 레코드 식별자를 사용하는 캐시 관리 방법들은 내용 검색 질의인 부분 부합 질의를 사용하는 환경에 사용할 수 없다. 또한, 유선 환경에서 사용하는 형식처럼, 서버와 클라이언트와의 일대일 통신 방법을 사용하는 캐시 관리 방법 역시, 무선 통신의 비용과 시스템의 확장성(scalability) 측면에서 현실적으로 사용이 불가능하다. 따라서, 이 장에서는 제안하는 캐시 관리 방법과 서버 데이타베이스의 전체를 방송하는 방법을 비교한다.

```
// Notation:
//   cache_predicate: the predicate describing client's cache,
//   invalid_predicate: the invalid predicate (i.e., l-bit cover predicates),
//   remainder_predicate_set: the set of remainder predicates

PROCEDURE Remainder_Generation_with_Intersection (cache_predicate)
{
        // 무효 predicate을 이용한 교차 연산 단계
        remainder_predicate_set = { }; // initially empty set
        for each invalid_predicate in P-CIR do {
                remainder_predicate = cache_predicate ∩ invalid_predicate;
                add remainder_predicate into remainder_predicate_set;
        }
        // 유효 이진 비트 열을 이용한 나머지 프레디킷 정제 단계
        for each valid_binary_bit_stream in P-CIR do {
                refine the remainder_predicate_set;
        }
}
```

그림 7 교차 연산을 이용한 나머지 프레디킷 생성 알고리즘

비교를 위하여 사용한 데이타는 TPC-C 벤치마크에서 정의한 Customer 테이블이다. 이 테이블은 16개의 애트리뷰트로 구성된 레코드가 30,000개 존재하며, 한 레코드 튜플의 크기는 686 바이트이다. 비교를 위하여 설정한 무선 통신 환경은 통신 대역폭이 14.4 Kbps인 IS-95 환경과(이것은 현재 우리가 사용하고 있는 PCS 시스템이다), 조만간 사용 가능할 것으로 보이는 IMT 2000 시스템의 2 Mbps 통신 대역폭이다. 프레디킷의 크기는 16개의 비트로 설정하였다. 이 값은 해싱 방법의 설계시 사용하는 load factor 0.5를 기준으로 설정한 값이다[10]. 이 환경을 사용하여, 제안하는 방법과 전체 데이타의 내용을 방송하는 경우의 CIR 방송 주기를 비교한 결과를 그림 8에 제시하였다.

방법 및 환경	전체 데이타 방송	프레디킷 기반 CIR 방송	
14.4 Kbps	약 3시간	1초(약 550개의 프레디킷)	2초(약 1100개의 프레디킷)
2 Mbps	약 1분 20초	1초(약 80000개의 프레디킷)	2초(약 160000개의 프레디킷)
비교	현실적으로 사용하기 어려운, 너무 긴 방송 주기를 필요로한다.	현실적으로 사용 가능하며, 또한 조정 가능한 방송 주기이다.	

그림 8 비교 결과

결과에서 알 수 있듯이, 전체 데이타를 모두 방송하는 경우에는, 너무 긴 방송 시간을 필요로 한다. 방송 시간이 길어지면, 그 만큼 캐시가 일관성이 보장되지 못하는 시간이 길어진다는 의미이다. 따라서, 캐시 관리 기법으로서의 기능을 제대로 하지 못한다. 반면에, 제안하는 프레디킷 기반 CIR 방송에서는 현실적으로 사용할 수 있는 방송 주기를 보이고 있으며, 시스템 환경에 따라, 적절한 방송 주기로 조정할 수 있다는 특징이 있다. 위 그림에서는 방송 주기를 1초로 설정하여 프레디킷을 선택한 예들이다. 물론, 데이타 전체를 방송하는 방법과 달리, 제안하는 방법에서는 캐시 일관성 유지를 위해, 추가적으로 서버에게 요청을 해서 데이타를 받아와야 하는 과정이 필요하다.

6. 결 론

연속 부분 부합 질의는 이동 컴퓨팅 및 무선 정보 시스템 분야에서 다양한 응용으로 사용될 수 있는 질의로서, 부분 부합 질의의 형태로 사용자에게 연속적으로 질의의 결과가 존재하여야 하는 질의이다. 본 논문에서는 이동 컴퓨팅 환경에서 널리 사용하는 캐시 무효화 방송 기법(CIR)을 사용할 때, 클라이언트의 캐시에 저장되는 부분 부합 질의 결과의 일관성 유지를 위한 캐시 관리 기법을 제안하였다.

연속 부분 부합 질의 처리에 관하여 기존에 연구된 결과는 없으며, 기존의 데이타 레코드 식별자를 기반으로 하는 캐시 기법은 내용 기반 질의인 부분 부합 질의의 처리에 적절하지 못하다. 이러한 문제점으로 인하여 본 논문에서는, 연속 부분 부합 질의의 표현을 위하여 프레디킷(Predicate) 개념을 사용하였으며, 이를 통해 클라이언트 캐시와 캐시 무효화 정보(Cache Invalidation Report: CIR)를 표현하였다.

그리고, 프레디킷 기반 CIR인 P-CIR을 이용한 캐시 일관성 방안에 대하여 기본적인 '억지 방법(the brute-force method)'을 살펴본 후, '빼기 연산 방법(the subtraction method)'과 '교차 연산 방법(the intersection method)'을 제안하였다. 억지 방법과 빼기 연산 방법이 계산 복잡도 측면에서 현실성이 떨어지는 문제점을 교차 연산 방법으로 해결할 수 있었다.

추후 연구 과제로서, 부분 부합 질의 형태가 아닌, 좀 더 일반적인 연속 질의 형태에 대한 캐시 관리 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] D. Barbara, "Mobile Computing and Databases-A Survey," IEEE Transactions on Data and Knowledge Engineering, 11(1), pp.108-117, 1999.

[2] T. Imielinski, et. al., "Data on Air: Organization and Access," IEEE Transactions on Data and Knowledge Engineering, 9(3), 1997.

[3] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching Strategies in Mobile Environments," In proceedings of ACM SIGMOD '94, pp.1-12, 1994.

[4] M. H. Kim and S. Pramanik, "Optimal File Distribution for Partial Match Retrieval," In proceedings of ACM SIGMOD '88, pp.173-182, 1988.

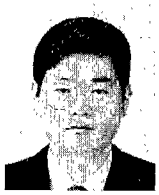
[5] J. Y. Lee, et. al., "A Data Scheduling Method for Reducing the Access Time of Partial Match Queries in the Wireless Data Broadcasting Environment," Journal of KISS: Databases, (to appear).

[6] D. Terry, et. al., "Continuous Queries over Append-Only Databases," In proceedings of ACM SIGMOD '92, pp.321-330, 1992.
[7] L. Liu, et. al., "Continual Queries for Internet Scale Event-Driven Information Delivery," IEEE Transactions on Data and Knowledge Engineering, 11(4), pp.610-628, 1999.
[8] C. C. Chang, et. al., "Symbolic Gray Code as a Perfect Multiattribute Hashing Scheme for Partial Match Queries," IEEE Transactions on Software Engineering, 8(3), pp. 234-248, 1982.
[9] K. A. S. Abdel-Ghaffar and A. E. Abbadi, "Optimal Disk Allocation for Partial Match Queries," ACM Transactions on Database Systems, 18(1), pp.132-156, 1993.
[10] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*. Addison-Wesley Publishing Company, 1973.

정 연 돈
1994 고려대학교 전산과학과 학사. 1996 한국과학기술원 전산학과 석사. 2000 한국과학기술원 전자전산학과 전산학전공 박사. 현재 한국과학기술원 정보전자연구소 Post-Doc. 연구원. 관심분야는 이동 컴퓨팅, 분산시스템, 데이타베이스 시스템

이 지 연
1993 동국대학교 전자계산학과 학사. 1996 한국과학기술원 전산학과 석사. 2001 한국과학기술원 전자전산학과 전산학전공 박사. 현재 현대정보기술(주) 기술센터 미래기술팀 선임연구원. 관심분야는 이동 컴퓨팅, 분산 시스템, 데이타 마이닝, 음성 정보 시스템

이 윤 준
1977년 서울대학교 계산통계학과 졸업. 1979년 한국과학기술원 전산학과에서 석사학위 취득. 1983년 France, INPGEN-SIMAG에서 박사학위 취득. 1983년 ~ 1984년 France, IMAG 연구원. 1984년 ~ 현재 한국과학기술원 전산학과 부교수. 1989년 MCC(미) 초빙연구원. 1990년 CRIN(불) 객원교수. 관심분야는 데이타베이스 시스템, 정보검색, 실시간 데이타베이스 등임.



김 명 호
1982년 서울대학교 컴퓨터공학과 학사. 1984년 서울대학교 컴퓨터공학과 석사. 1989년 MICHIGAN 주립대 연구원. 1989년 ~ 1993년 KAIST 부교수. 1993년 ~ 현재 KAIST 부교수. 1992년 ~ 1993년 개방형 컴퓨터 통신 연구회(OSIA) 분산 트랜잭션 처리 분과위(TG-TP)의장. 1993년 ~ 1994년 한국통신기술협회(TTA) 분산 트랜잭션처리 실무 위원회 의장. 관심분야는 분산데이타베이스, 분산트랜잭션, 멀티미디어 데이타베이스